Towards a standard-based domain-specific platform to solve machine learning-based problems

Vicente García-Díaz, Jordán Pascual Espada, B. Cristina Pelayo G-Bustelo, and Juan Manuel Cueva Lovelle

Department of Computer Science, University of Oviedo, Oviedo, Spain

***Abstract* — Machine learning is one of the most important subfields of computer science and can be used to solve a variety of interesting artificial intelligence problems. There are different languages, framework and tools to define the data needed to solve machine learning-based problems. However, there is a great number of very diverse alternatives which makes it difficult the intercommunication, portability and re-usability of the definitions, designs or algorithms that any developer may create. In this paper, we take the first step towards a language and a development environment independent of the underlying technologies, allowing developers to design solutions to solve machine learning-based problems in a simple and fast way, automatically generating code for other technologies. That can be considered a transparent bridge among current technologies. We rely on Model-Driven Engineering approach, focusing on the creation of models to abstract the definition of artifacts from the underlying technologies.**

***Keywords* — Domain-Specific Language, Model-Driven Engineering, Integrated Development Environment, Machine Learning, Artificial Intelligence, Xtext**

I. Introduction

Artificial Intelligence (AI) refers to the "intelligence" provided by software included in some machines [1]. It is also a field of study commonly defined as the design of intelligent agents, which perceives their environment and takes actions that maximize their possibility of success [2]. The general problem of creating intelligence can be divided into different sub problems: 1) deduction and reasoning; 2) knowledge representation; 3) planning; 4) social intelligence; 5) natural language processing; 6) perception; 7) motion and manipulation; 8) long-term goals; or 9) machine learning.

Machine learning is one of the most important applications of artificial intelligence that evolved from the study of pattern recognition and computational learning theory. The goal is to create and study algorithms that are capable of leaning from data and make predictions on its basis [3].

Designing and implementing algorithms for machine learning is not a trivial tasks. M. Mitchell stated that a computer program is said to learn from experience E with respect to some class of tasks T and performance P, if its performance with tasks T, as measured by P, is improved with experience E [4].

In addition, machine learning is closely related to many other areas such as computational statistics for prediction-making or mathematical optimization, making the number of people involved in using related techniques very diverse with different backgrounds. Thus, there is a large number of solutions using different approaches to deal with machine learning-based problems. For example, Encog is a machine learning framework available for Java, .NET and C++ programmers [5], and Weka is a workbench that contains a group of graphical tools for data analysis and predictive modeling [6]. Moreover, there are also used General-Purpose Languages (GPL) such as Python, that is suitable for students and is one of the most popular introductory programming languages [7]. On the other hand, Domain-Specific Languages (DSL) [8] such as R, are also used for machine learning tasks [9] and their relevance continue growing.

However, although there is a great amount of solutions to deal with machine learning-based problems, all of them seem to be difficult to be used by no-experts programmers or require users to learn different technologies or applications that make the knowledge they have about a tool or platform virtually useless when they need to work with another one, when circumstances require it. This is even more problematic when the solution should be done programmatically for better control and adaptation.

Hence, different tools and software development approaches continuously appear in the software engineering field, trying to abstract the development from specific platforms or technologies (e.g., virtual machines, APIs, frameworks, etc.). It is widely considered that the Model-Driven Engineering (MDE) approach, with which the level of abstraction of developments is increased through the use of models, it is a step forward in the development of software [10], since developments are being benefited from the advantages provided by MDE (e.g., in García-Díaz et al. [11] food traceability systems for different clients are created in a quick and dynamic way).

MDE is based on the use of models, which conform to a single domain-based metamodel, which in turn are defined based on a common meta-metamodel, root of all the elements of any software development. That idea makes up the architecture of four layers defined in the Model-Driven Architecture (MDA) standard [12]. The common base allows for a wide range of supported environments and tools working together. As a result, if a metamodel for a specific knowledge domain is defined (e.g., food traceability or machine learning), it would be possible to create a DSL based on MDE tools [13], designed only to define the important specific items (e.g., food manufacturing processes or features of neural networks). Internally, the use of standard-based modeling technologies allows direct and automatic transformations to different formats or platforms defined by different software manufacturers. There are a variety of research in MDE that serve to advance in the systematic use of DSLs. For example, Cueva et al. work on bringing together the MDE approach and the Internet of Things field creating languages for automatic vehicle data capture [14][15] or García.-Díaz et al. work on improvement match algorithms for performing further operations with models [16].

The main aim of this paper is to take the first step towards the creation of a standard-based platform for defining and abstracting machine learning-based solutions in a simple and common way. Internally, definitions are automatically transformed into different

DOI: 10.9781/ijimai.2015.351

languages or platforms. Thus, the specific goals are:

1. Identify the basic elements that a representation of a language for solving machine learning-based problems must possess.

2. Create a DSL to define machine learning-based solutions. We call it AiDSL.

3. Allow automatic transformation of definitions made with AiDSL to any other platform or system.

4. Provide an Integrated Development Environment (IDE) to work with AiDSL. We call it AiIDE.

5. Study the advantages of the proposal by a comparison with other alternatives.

The remainder of this paper is structured as follows: in Section 2, we present a description of the relevant state of the art (goal [1]); in Section 3, we describe our proposal (goals [2-4]); in Section 4, we discuss a comparison of the proposal with other alternatives (goal [5]) and finally, in Section 5, we indicate our conclusions and future work to be done.

II. Background

There are a large number of approaches to deal with machine learning-based problems such as: 1) decision tree learning; 2) association rule learning; 3) artificial neural networks; 4) inductive logic programming; 5) support vector machines; 6) clustering; 7) Bayesian networks; 8) reinforcement learning; 9) representation learning; 10) similarity and metric learning; 10) sparse dictionary learning; or 11) generic algorithms.

In this work we focus on Artificial Neural Networks (ANN), that have been used to solve a great variety of problems that are difficult to solve using other techniques [17]. They can be defined as statistical learning models inspired by biological neural networks. Typically, there are presented as collections of interconnected neurons, sending messages each other. Each neuron has numeric weights that can be set using different algorithms, being them adaptive to inputs, or what it is the same, allowing them to learn.

Regarding ANNs, the Feedforward neural network was the first and the simplest type of ANN formulated. In such a type of network the information moves only in one direction. In this work, we focus on the Feedforward artificial neural network, although there are some other such as Elman Neural Network or the Jordan Neural Network, interesting depending on the type of problem to be solved.

Fig. 1 shows a small example of an artificial neural network, which can be decomposed into different layers, containing each a specific number of neurons with similar properties.

- Input layer. Typically it has one neuron for each attribute that the network will use for obtaining different kinds of solutions (e.g., classification, regression or clustering). In the example, I1 and I2 are input neurons included in the input layer.
- Output layer. It provides the output after all previous layers have processed the input. In the example, O1 is the only output neuron that is included in the output layer.
- Hidden layers. They are inserted between input and output layers and are used to better produce the expected output for the given input readjusting weights. In the example, H1 and H2 are hidden neurons contained in the only hidden layer shown.

In addition, there are also bias neurons that can be inserted in the input and hidden layers as desired (B1 and B2 in the example). They are very similar to the hidden neurons but are a special kind that allow the neural network to learn patterns more effectively, always returning the maximum value, without receiving any input.

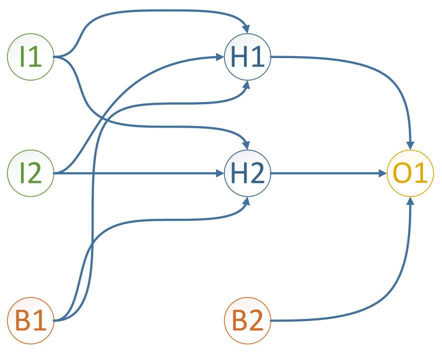


Fig 1. Artificial neural network (example)

From the point of view of classification, there are typically three broad categories into machine learning: 1) supervised learning. The algorithm is trained with example inputs and outputs. For example, Carneiro proposes a method for semantic image annotation and retrieval [18]; 2) unsupervised learning. The algorithm is not trained with examples but other techniques such as generic algorithms help to find the correct solution. For example, Kattan et al. predict the position of any particular target event in a time series [19]; and 3) reinforcement learning. The algorithm learns its behaviour based on feedback from the environment. For example, Gosavi uses reinforcement learning for control optimization [20]. There can be other definitions such as semi-supervised learning, learning to learn, developmental learning and even other classifications like for example depending on the expected kind of output.

ANNs are linked to a large amount of different type of scenarios. For example: 1) any kind of control system [21]; 2) autonomous navigation of robots [22]; 3) pattern recognition [23]; 4) forecasting [24]; or estimation of heating loads of buildings [25].

Those ANNs-based problems can be formulated using different technologies such as:

- Encog for Java, .NET or C++ [5], supporting different algorithms such as hidden Markov Models, vector machines, Bayesian networks and neural networks.
- AForge.NET for .NET [26], designed for developers and researchers in the fields of computer vision and artificial intelligence.
- The SHOGUN machine learning toolbox [27], with interfaces for MATLAB, R, Octave and Python, apart from a stand-alone command line interface.
- Apache Mahout [28], providing free implementations of scalable machine learning algorithms, using Java libraries for common operations.
- Many others such as Weka [6], Spark MLlib or ConvNetJS.

The main problem is that there are not bridges among the previous technologies, so users are highly dependent on the underlying technology. In addition, when programming is needed, software libraries that are provided does not have a high level of abstraction since they usually are designed for GPLs with no semantics in the language linked to the type of problems to be solved. That forces users to have both high programming and machine learning skills to make use of them.

III. Overview of the System

To design the prototype, we used the MDE development approach, raising the level of abstraction of software engineering. Specifically, we have used the tools built on the Eclipse Modeling Project (EMP) [29], offering one of the most accepted implementations of the standards promoted by the Object Management Group (OMG) [30].

OMG is the organization that has driven the development of MDE through the creation of the set of standards enclosed in the Model-Driven Architecture (MDA) [12] specification, for the effective and efficient work under the MDE paradigm. It is usually carried out by creating DSLs tied to a specific domain of knowledge and generating artifacts for different platforms and manufactures.

Decision in favor of a new DSL is usually not easy because it is much less expensive (both in time and cost) to adopt an existing DSL if available or even to use a GPL such as Java or C#. For that, there are mainly only two reasons why it is worth creating a new DSL [31]: 1) improved software economics, giving some authors as a reference point three developments [32] to obtain a positive return on investment; and 2) allow that people with less domain and programming expertise to develop software, even end-users with some domain, but no programming expertise [33][34].

Since our goals are compatible with both criteria, we have created a new DSL based on common elements used to describe the structural part of a feedforward neural network. To that end, we have used the Xtext framework [35], which allows the creation of both GPLs and DSLs in a relatively easy way [36]. From a grammar and some other definitions, it is possible, for example, to get a working parser and linker and also a complete Eclipse-based Integrated Development Environment [37]. Xtext also provides several mechanisms through which you can configure different aspects of languages such as validations of code, syntax highlighting, proposals to developers, code formatting or even generating artifacts through programs implemented with the programming languages defined with Xtext.

A. AiDSL

Next, there is a snippet of the context-free Xtext grammar used as the basis of the AiDSL language.

```
AI:
   neuralNetworks+=NeuralNetwork*;

NeuralNetwork:
   “neuralNetwork” name=ID “{“
      “neurons” “{“
         “input” “{“
            inputLayer = InputLayer
         “}”
         “hidden” “{“
            hiddenLayer = HiddenLayer
         “}”
         “output” “{“
            outputLayer = OutputLayer
         “}”
      “}”
      “training” “{“
         “input:” trainingInput = FloatsColection
         “output:” trainingOutput = FloatsColection
         (“type:” trainingType = TrainingType)?
(“errorThreshold:” trainingErrorThreshold = Float)?
         (“result:” trainingResult = Result)?
      “}”
      “data” “{“
         “input:” dataInput = FloatsColection
         (“result:” dataResult = Result)?
      “}”
   “}”
;

InputLayer:
   “size:” size = INT
   bias ?= “bias”?
;

HiddenLayer:
   “size:” size = INT
   bias ?= “bias”?
   (“activation:” activation = Activation)?
;

OutputLayer:
   “size:” size = INT
   (“activation:” activation = Activation)?
;

enum Activation:
   BiPolar |
   Competitive |
   HyperbolicTangent |
   Linear |
   LOG |
   Sigmoid |
   SoftMax
;

enum TrainingType:
   Backpropagation |
   QuickPropagation = “QPROP”|
   LevenbergMarquardt = “LMA”|
   ManhattanUpdateRule |
   ResilentPropagation = “RPROG” |
   ScaledConjugateGradient = “SCG”
;

FloatsColection: (‘[’ Floats ‘]’)+;
Floats: Float(‘,’Float)*;
Float: INT(‘.’INT)?;

enum Result:
   Console |
   None
;
```

With this grammar, neural networks can be created indicating information about the input, the hidden and the output layers (e.g., number of neurons, presence of bias neurons and the activation mode). Activation functions are attached to layers and are needed to scale data output from a layer. There are different activation functions available (users can select among different functions depending on the case: bipolar, competitive, hyperbolic tangent, linear, log, sigmoid or softmax). Depending on the selection, network behavior will be different. As we focus on supervised learning, we define the way users can introduce training data with inputs and expected outputs and the algorithm used for training the system (Back propagation, Quick propagation, Levenberg Marquardt, Manhattan update rule, Resilent propagation or Scaled conjugated gradient), also depending on each particular problem. More information about the theoretical basis for designing neural networks could be found for example in Haykin [38].

The Xtext-based grammar is transformed internally into an ANTLR grammar [39] to implement the lexer (lexical analysis) and the parser (syntactic analysis) that is used when a programming language is being defined. In addition, it also generates all the necessary infrastructure to create the Abstract Syntax Tree (AST) to perform a semantic analysis on the language elements. The iteration through the tree is performed using model-based technologies, particularly the Eclipse Modeling Framework [40], which serves to ensure interoperability of the generated DSL with many other model-based existing tools such as the tools defined in the Eclipse Modeling Project [29] to help improve software development productivity.

The definition of such a grammar leads to a metamodel for the domain that is automatically generated. This metamodel makes programs that are made based on it to follow a formal definition that allows to use any tool that is compatible with all standards promoted by the MDA such as interoperability, reusability and portability, opening a wide range of possibilities. Thus, every time a new program with AiDSL is created, a model that conforms to the proposed metamodel is instantiated, following its rules and offering a formalism that makes it very easy to perform different tasks such as validation, storing or generation of artifacts. The rules are defined in general terms based on the metamodel of the language, not for each individual case, that is, not for each model obtained during the development, which facilitates the process.

B. *Transformations from AiDSL to other technologies*

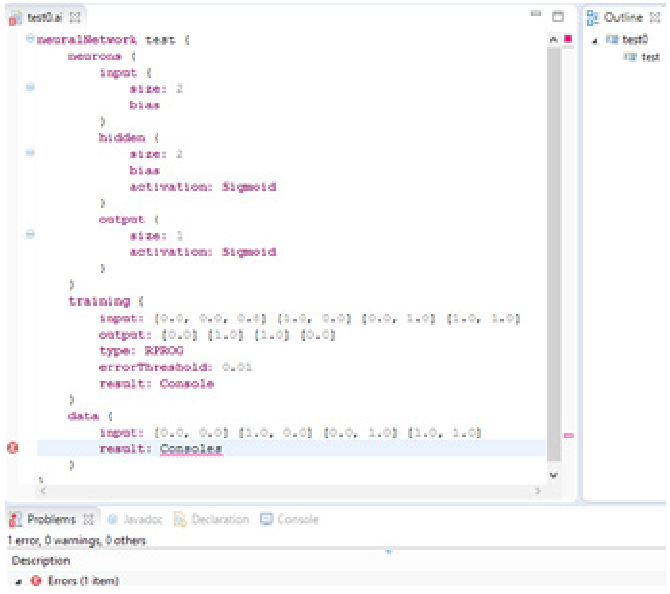
The code below shows a fragment of the template that is used to generate artifacts from any of the models defined using AiDSL, based on its grammar. In this example, programmed with the Xtend language, the generation is focused on the Encog machine learning framework [5] but with other templates, code for other platforms could be generated without further changes. In addition, it could be possible to directly interpret models without the need of focusing on any platform. The idea of this approach is to generate from a model, easily and automatically, the code for different architectures or platforms (it would only be necessary to add new templates). That would be a key step to benefit from all the advantages of integration and reuse offered by the MDE approach (e.g., the use of common repositories and version control systems for models).

```
@SuppressWarnings("unused")
public class «n.name.toFirstUpper» {
   public     static     double     trainingInput[][]     =
«n.trainingInput.floatsColection»;
   public     static     double     trainingOutput[][]     =
«n.trainingOutput.floatsColection»;
   public static double dataInput[][] =
«n.dataInput.floatsColection»;

   public void run() {
      BasicNetwork network = new BasicNetwork();
      network.addLayer(new      BasicLayer(null,      «IF
n.inputLayer.bias      ==      true»true«ELSE»false«ENDIF»,
«n.inputLayer.size»));
      network.addLayer(new BasicLayer(new «n.hiddenLayer.
activation.toString.activation»(), «IF n.hiddenLayer.bias
== true»true«ELSE»false«ENDIF», «n.hiddenLayer.size»));
      network.addLayer(new BasicLayer(new «n.outputLayer.
activation.toString.activation»(), false, «n.outputLayer.
size»));
      network.getStructure().finalizeStructure();
      network.reset();

      MLDataSet          trainingSet          =          new
BasicMLDataSet(trainingInput, trainingOutput);
      «trainingType(n.trainingType)»

      int epoch = 1;
      do {
         train.iteration();
         «IF n.trainingResult == Result.CONSOLE»
         System.out.println("Epoch #" + epoch + " Error:"
+ train.getError());
         «ENDIF»
         epoch++;
      }             while(train.getError()             >
«n.trainingErrorThreshold»);
      train.finishTraining();

      MLDataSet  dataSet  =  new  BasicMLDataSet(dataInput,
null);
      «IF n.dataResult == Result.CONSOLE»
      System.out.println("Neural Network Results:");
      «ENDIF»
      for(MLDataPair pair: dataSet ) {
         final   MLData   output   =   network.compute(pair.
getInput());
         «IF n.dataResult == Result.CONSOLE»
         System.out.println(pair.getInput() +
               " => actual=" + output);
         «ENDIF»
      }

      Encog.getInstance().shutdown();
   }
}
'''
```

C. *AiIDE*

Based on the Xtext architecture, some of the features included in the development environment called AiIDE are:

- Custom syntax-highlighting to distinguish the different elements of the language (e.g., keywords, comments or variables). This is done by implementing the Xtext interfaces IHighlightingConfiguration and ISemanticHighlightingCalculator.
- Content assistant to help the developer to write code faster and more efficiently through the use of the auto-complete functionality (extending the TerminalsProposalProvider class).
- Static validation of the language elements to detect syntactic and semantic issues (extending the AbstractDeclarativeValidator class).
- Suggestions for fixing errors or problems identified in the code (extending the DefaultQuickfixProvider class).
- Templates that allow developers to reduce the learning curve for typical operations.
- Formatting the code through a feature called code beautifier to distribute it properly and promote its maintenance (extending the AbstractDeclarativeFormatter class).
- Outline view fully configurable to both the elements that appear and text or icons attached to them (extending the DefaultObjectLabelProvider class).



```
neuralNetwork test {
    neurons {
        input {
            size: 2
            bias
        }
        hidden {
            size: 2
            bias
            activation: Sigmoid
        }
        output {
            size: 1
            activation: Sigmoid
        }
    }
    training {
        input: [0.0, 0.0, 0.0] [1.0, 0.0] [0.0, 1.0] [1.0, 1.0]
        output: [0.0] [1.0] [1.0] [0.0]
        type: RPROG
        errorThreshold: 0.01
        result: Console
    }
    data {
        input: [0.0, 0.0] [1.0, 0.0] [0.0, 1.0] [1.0, 1.0]
        result: Consoles
    }
}
```

1 error, 0 warnings, 0 others
Description
Errors (1 item)

Fig. 2. AiDE working

Fig. 2 is a screenshot of the environment when a model is being created. It can be seen different features. For example, the syntax-highlighting for different elements (e.g., neurons, bias, training, etc.), the static validation marking a result type as not valid because

TABLE 1
GUIDELINES FOR A BETTER QUALITY AND A BETTER ACCEPTANCE AMONG ITS USERS

GUIDELINE	ACCOMPLISHMENT
Language purpose	
Identify language uses early	The language is used mainly for documentation of knowledge and code generation
Ask questions about uses	Any person with interest in defining neural networks will be able to model with the language
Make your language consistent [42]	It is consistent with the sole idea of defining and creating neural networks at this point
Language realization	
Decide carefully whether to use graphical or textual realization	Textual realization based on the advantages noted by Groenniger et al. [43]: 1) need of less space to display the same information; 2) more efficient creation of code; 3) easier integration with other languages; 4) more speed and quality of the formatting; 5) platform and tool independency; and 6) better version control support. Besides, graphical realizations provide a better overview and ease the understanding of models [41], but in a very close and specific domain like neural networks, we believe that the advantages of textual languages are more important
Compose existing languages where possible	Instead of starting from scratch, we have used the entire ecosystem of tools provided by the Eclipse Modeling Project, specifically Xtext, a DSL to define other DSLs, which relies heavily on the use of the Xtend language, an extension of the Java language, primarily intended to support the creation of DSLs (e.g., validations and code generation)
Reuse existing language definitions	To create AiDSL we have used the core grammar of Xtext as a basis to avoid redefining elements already defined previously
Reuse existing type systems	Related to the previous point, we have reused the core data types defined by the creators of Xtext with the aim of reusing the existing knowledge
Language content	
Reflect only the necessary domain concepts	It only contains the basic elements needed to generate code in the different target formats, so no extra domain concept is added
Keep it simple	With a small number of elements, simple syntax and reduced domain of knowledge, we think that the language is easier than other alternatives. The quantitative analysis also suggests the same idea
Avoid unnecessary generality	Due to the close domain of the language, we did not include the generalization concept, meeting with the principle of designing only what is necessary
Limit the number of language elements	The language is small, having only 13 domain-specific keywords (e.g., Java has 50 generic keywords and C# even more)
Avoid conceptual redundancy	Each fact can only be described in a unique way, avoiding redundancy
Avoid inefficient language elements	Each element is needed for clarity and used with the only purpose of allowing the generation of the final code, so there are no inefficient language elements
Concrete syntax	
Adopt existing notations domain experts use [44]	Neural networks are usually defined using a graph-based structure with inputs, outputs an intermediate nodes or states
Use descriptive notations	The language has a small number of keywords with syntax highlighting and code completion support. In addition, frequently-used symbols in other languages such as =, { or } maintain their semantics
Make elements distinguishable	Keywords, different syntax highlighting and an outline view are used to make elements distinguishable
Use syntactic sugar appropriately	We avoid syntactic sugar since we think that in a small DSL expressing the same concepts in different ways can be counterproductive, confusing users and hindering validation and code generation unnecessarily
Permit comments [45]	Support for common types of comments: single-line comments (//) and multi-line comments (/*..*/)
Provide organizational structures for models	Organizational structures such as packages are important for complex systems. However, to keep the language simple, we intend to have the definition of the set of neural networks in the same organizational structure
Balance compactness and comprehensibility	The quantitative analysis suggests that this approach may require less elements than other approaches. However, since it is a DSL with concrete semantics for the domain, it is even more comprehensible
Use the same style everywhere	All the elements of the language have the same look-and-feel and we do not embed any external language that can difficult the understanding of the language by using another syntax
Identify usage conventions	Based on an ANTLR grammar we define typical usage conventions including notation of identifiers, order of elements or type of comments
Abstract syntax	
Align abstract and concrete syntax	We took into account the three principles mentioned in Karsai et al. [41]: 1) elements that differ in the concrete syntax also have different abstract notations (e.g., input layer and type of learning are based on different metaclasses); 2) elements that have a similar meaning can be internally presented by reusing concepts of the abstract syntax (e.g., the FloatsColection rule for indicating inputs and outputs has been created using two int values along with other literals such as "[", "]" or "."); and 3) the abstract notation should not depend on the context an element is used but only on the element itself
Prefer layout which does not affect translation from concrete to abstract syntax	To simplify the usage of the DSL, the layout of the models does not affect the semantics. For example, modelers can use tabs, spaces or line breaks whenever they want. However AiIDE provides the feature called code beautifier, also provided by some environments to automatically place the language elements in a way easily understandable for most potential users
Enable modularity [46]	It is possible to decompose the code into smaller files, referencing them from other files. However, for this small language, we think that it is not necessary and it may unnecessarily increase the difficulty of use
Introduce interfaces	Interfaces are an important feature in complex systems, increasing flexibility and maintenance. However, we did not need them in our DSL because it is a simple declarative language

instead of Console, the programmer typed Consoles as the way to show the output, and the outline view showing a summary of the elements that are being used (in the example just a neural network inside the file).

In addition, it is possible to perform other customizations such as specifying the scope of the variables of the language. Thus, the AiIDE is a full-fledged development environment integrated in the Eclipse platform with the resulting advantages it provides (e.g., well-known and proven platform for developers, large amount of tools and plug-ins, open environment, etc.).

IV. Evaluation

The sections below are dedicated to a qualitative and quantitative study to show the characteristics of AiIDE and AiDSL, justifying the design and the need for its creation.

A. *Qualitative analysis*

To achieve a better quality of the language and the environment design and a better acceptance among its users, Karsai et al. [41] have proposed some guidelines largely based on their experience in developing languages as well as relying on existing guidelines on programming and modeling languages. Table 1 serves to verify that these guidelines are met.

B. *Quantitative analysis*

In this section we briefly evaluate the AiDSL language. We obtain a quantitative measurement that allows us to evaluate the main objective of our proposal; simplify and make more agile the definition of machine learning-based solutions.

In this first step of the development we are going to do a brief comparison between the definitions of two different neural networks using both AiDSL and the Encog framework. Since with AiDSL it is possible to automatically generate code for Encog and any other technology, if the syntax used by AiDSL is more compact, then it can clearly be seen as advantageous over other languages or frameworks. The measured aspects in the code and the structure are the ones below:

- Code lines: it refers to the number of lines of information needed to define the neural networks in each case.
- Words: number of words used.
- Characters: number of characters, spaces included.

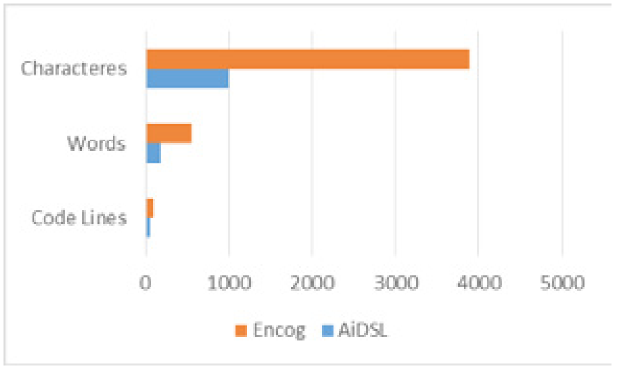


Fig. 3. Comparing AiDSL and Encog working with two neural networks

In the obtained results of the analysis (Fig. 3), we can observe that with AiDSL we require much less code lines (56 vs 102), words (180 vs 549) and characters (996 vs 3895) to define the same information than with the Encog framework. For the measurements, we defined two neural networks with a number of input, hidden and output neurons, activation method, training information and type of output expected. After we defined it with AiDSL the AiIDE automatically generated the code that should be necessary if we have worked directly with the Encog framework.

V. Conclusions and Future Work

In this paper we have presented the first version of a language for defining neural networks (AiDSL) and a development environment to facilitate working with those networks (AiIDE). This has been done by identifying basic elements that are useful to define the important aspect of any artificial neural network. In addition, we have defined mappings for transforming models made with AiDSL to the code that should be used if we worked with the Encog framework instead, and created the basis to do the same with other different popular frameworks (e.g., Weka), which favors the development and increases productivity and interoperability among systems. Finally, it the use of AiDSL through the AiIDE is easier than the manual and specific handling of other frameworks with identical purposes. Of course, both AiDSL and AiIDE are prototypes with limited scope and popular frameworks such as Encog or Weka offer many more features.

From the point of view of computer science, the focus of this paper could be set embedded in this category: Artificial Intelligence → Machine Learning → Neural Network → Feedforward Neural Network → Supervised learning. Further works will focus on other areas while they will delve into supervised learning.

Future work will be to improve and adapt both AiIDE and AiDSL with new frameworks and features to define neural networks. Finally, we will perform a usability study with real users for quantifying how simple, easy and intuitive is our proposal for them. The idea is to work with people with different profiles and ask them to define several neural networks using different techniques. That way, we will observe, among other things, the efficiency, the learning curve and the number of errors that are performed during the tasks.

References

[1] S. Russell, P. Norvig, and A. Intelligence, "A modern approach," Artif. Intell. Prentice-Hall, Egnlewood Cliffs, vol. 25, 1995.
[2] D. Poole, A. Mackworth, and R. Goebel, Computational Intelligence: A Logical Approach. Oxford, UK: Oxford University Press, 1997.
[3] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "An overview of machine learning," in Machine learning, Springer, 1983, pp. 3–23.
[4] T. M. Mitchell, "Machine learning. WCB." McGraw-Hill Boston, MA:, 1997.
[5] H. Jeff, "Programming Neural Networks with Encog3 in Java," 2011.
[6] G. Holmes, A. Donkin, and I. H. Witten, "Weka: A machine learning workbench," in Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on, 1994, pp. 357–361.
[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and others, "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.
[8] A. Van Deursen, P. Klint, and J. Visser, "Domain-Specific Languages: An Annotated Bibliography.," Sigplan Not., vol. 35, no. 6, pp. 26–36, 2000.
[9] B. Lantz, Machine learning with R. Packt Publishing Ltd, 2013.
[10] S. Kent, "Model driven engineering," in Integrated Formal Methods, 2002, pp. 286–298.
[11] V. García-Díaz, J. Tolosa, B. G-Bustelo, E. Palacios-González, Ó. Sanjuan-Martínez, and R. Crespo, "TALISMAN MDE Framework: An Architecture for Intelligent Model-Driven Engineering," in Distributed Computing Artificial Intelligence Bioinformatics Soft Computing and Ambient Assisted Living, vol. 5518, S. Omatu, M. Rocha, J. Bravo, F. Fernández, E. Corchado, A. Bustillo, and J. Corchado, Eds. Springer

Berlin / Heidelberg, 2009, pp. 299–306.
[12] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, "Model-driven architecture," in Advances in Object-Oriented Information Systems, Springer, 2002, pp. 290–297.
[13] E. P. González, H. F. Fernández, V. G. Díaz, B. C. P. G. Bustelo, J. M. C. Lovelle, and O. S. Martínez, "General purpose MDE tools," IJIMAI, vol. 1, no. 1, pp. 72–75, 2008.
[14] G. C. Fernandez, J. P. Espada, V. G. Díaz, and M. G. Rodríguez, "Kuruma: the vehicle automatic data capture for urban computing collaborative systems," Int. J. Interact. Multimed. Artif. Intell., vol. 2, no. 2, pp. 28–32, 2013.
[15] G. Cueva-Fernandez, J. P. Espada, V. García-Díaz, R. G. Crespo, and N. Garcia-Fernandez, "Fuzzy system to adapt web voice interfaces dynamically in a vehicle sensor tracking application definition," Soft Comput., pp. 1–14, 2015.
[16] V. García-Díaz, B. C. P. G-Bustelo, O. Sanjuán-Martínez, E. R. N. Valdez, and J. M. C. Lovelle, "MCTest: towards an improvement of match algorithms for models," IET Softw., vol. 6, no. 2, p. 127, Apr. 2012.
[17] B. Yegnanarayana, Artificial neural networks. PHI Learning Pvt. Ltd., 2009.
[18] G. Carneiro, A. B. Chan, P. J. Moreno, and N. Vasconcelos, "Supervised learning of semantic classes for image annotation and retrieval," Pattern Anal. Mach. Intell. IEEE Trans., vol. 29, no. 3, pp. 394–410, 2007.
[19] A. Kattan, S. Fatima, and M. Arif, "Time-series event-based prediction: An unsupervised learning framework based on genetic programming," Inf. Sci. (Ny)., 2015.
[20] A. Gosavi, "Control Optimization with Reinforcement Learning," in Simulation-Based Optimization, Springer, 2015, pp. 197–268.
[21] W. T. Miller, P. J. Werbos, and R. S. Sutton, Neural networks for control. MIT press, 1995.
[22] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," Neural Comput., vol. 3, no. 1, pp. 88–97, 1991.
[23] C. G. Looney, Pattern recognition using neural networks: theory and algorithms for engineers and scientists. Oxford University Press, Inc., 1997.
[24] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," Int. J. Forecast., vol. 14, no. 1, pp. 35–62, 1998.
[25] S. A. Kalogirou, "Artificial neural networks in renewable energy systems applications: a review," Renew. Sustain. energy Rev., vol. 5, no. 4, pp. 373–401, 2001.
[26] [26] A. Kirillov, "AForge .NET framework," 2010-03-02)[2010-12-20]. http://www. aforgenet. com. 2013.
[27] [27] S. Sonnenburg, G. Rätsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. de Bona, A. Binder, C. Gehl, and V. Franc, "The SHOGUN machine learning toolbox," J. Mach. Learn. Res., vol. 11, pp. 1799–1802, 2010.
[28] [28] S. Owen, R. Anil, T. Dunning, and E. Friedman, Mahout in action. Manning, 2011.
[29] R. C. Gronback, Eclipse modeling project: a domain-specific language (DSL) toolkit. Pearson Education, 2009.
[30] O. M. G. CORBA and I. Specification, "Object Management Group." Joint revised submission OMG document orbos/99-02-, 1999.
[31] M. Mernik, J. Heering, and A. M. Sloane, "When and How to Develop Domain-specific Languages," ACM Comput. Surv., vol. 37, no. 4, pp. 316–344, 2005.
[32] K. Schmid and M. Verlage, "The economic impact of product line adoption and evolution," IEEE Softw., vol. 19, no. 4, pp. 50–57, 2002.
[33] B. A. Nardi, A small matter of programming: perspectives on end user computing. MIT press, 1993.
[34] M. Voelter, "A Catalog of Patterns for Program Generation.," in EuroPLoP, 2003, pp. 285–320.
[35] S. Efftinge and M. Völter, "oAW xText: A framework for textual DSLs," in Workshop on Modeling Symposium at Eclipse Summit, 2006, vol. 32, p. 118.
[36] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, 2010, pp. 307–309.
[37] J. desRivieres and J. Wiegand, "Eclipse: A platform for integrating development tools," IBM Syst. J., vol. 43, no. 2, pp. 371–383, 2004.
[38] S. Haykin and N. Network, "A comprehensive foundation," Neural Networks, vol. 2, no. 2004, 2004.
[39] T. J. Parr and R. W. Quong, "ANTLR: A predicated-LL (k) parser generator," Softw. Pract. Exp., vol. 25, no. 7, pp. 789–810, 1995.
[40] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, EMF: eclipse modeling framework. Pearson Education, 2008.
[41] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, and S. Völkel, "Design guidelines for domain specific languages," arXiv Prepr. arXiv1409.2378, 2014.
[42] B. Meyer, "Eiffel-The Language Prentice Hall," Englewood Cliffs, NJ, 1992.
[43] H. Grönninger, H. Krahn, B. Rumpe, M. Schindler, and S. Völkel, "Textbased modeling," arXiv Prepr. arXiv1409.6623, 2014.
[44] D. Wile, "Lessons learned from real DSL experiments," in System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on, 2003, p. 10–pp.
[45] R. S. Scowen and B. A. Wichmann, "The definition of comments in programming languages," Softw. Pract. Exp., vol. 4, no. 2, pp. 181–188, 1974.
[46] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations," in Proceedings of the 33rd International Conference on Software Engineering, 2011, pp. 411–420.

Vicente García-Díaz is an Associate Professor in the Computer Science Department of the University of Oviedo. He has a PhD from the University of Oviedo in computer engineering. His research interests include Domain-Specific Languages, Model-Driven Engineering, Business Process Management, Machine Learning, Internet of Things and eLearning.
Contact address is Computer Science Department, University of Oviedo. Edificio de la Facultad de Ciencias. C/ Calvo Sotelo s/n. 33007 Oviedo (Asturias, España); e-mail: garciavicente@uniovi.es

Jordán Pascual Espada is a research scientist at Computer Science Department of the University of Oviedo. Ph.D. from the University of Oviedo in Computer Engineering B.Sc. in Computer Science Engineering and a M.Sc. in Web. He has published several articles in international journals and conferences, he has worked in several national research projects. His research interests include the Internet of Things, exploration of new applications and associated human computer interaction issues in ubiquitous computing and emerging technologies, particularly mobile and Web applications.
Contact address is Computer Science Department, University of Oviedo. Edificio de la Facultad de Ciencias. C/ Calvo Sotelo s/n. 33007 Oviedo (Asturias, España); e-mail: pascualjordan@uniovi.es

Cristina Pelayo García-Bustelo is a lecturer in the Computer Science Department of the University of Oviedo. She has a PhD from the University of Oviedo in computer engineering. Her research interests include object-oriented technology, Web engineering, eGovernment, modeling software with BPM, DSL and MDA.
Contact address is Computer Science Department, University of Oviedo Edificio de la Facultad de Ciencias. C/ Calvo Sotelo s/n. 33007 Oviedo (Asturias, España); e-mail: crispelayo@uniovi.es

Juan Manuel Cueva Lovelle became a mining engineer from Oviedo Mining Engineers Technical School in 1983 (Oviedo University, Spain). He has a PhD from Madrid Polytechnic University, Spain (1990). From 1985 he has been a professor at the languages and computers systems area in Oviedo University (Spain), and is an ACM and IEEE voting member. His research interests include object-oriented technology, language processors, human-computer interface, Web engineering, modeling software with BPM, DSL and MDA.
Contact address is Computer Science Department, University of Oviedo. Edificio de la Facultad de Ciencias. C/ Calvo Sotelo s/n. 33007 Oviedo (Asturias, España); e-mail: cueva(at)uniovi.es